COBB

INSIDE DOS

Tips & techniques for MS-DOS & PC-DOS Versions 5 & 6

February 1994
Vol. 5 No. 2

VERSION 6.x

Compressing a floppy disk with DoubleSpace

If you're like many DOS users, you've approached the idea of using DoubleSpace to increase your hard disk capacity with a lot of caution. You might not even want to consider compressing a floppy disk—but you should.

Compressing a floppy disk with DoubleSpace isn't much different than compressing your hard drive. In many ways, it's even easier. In this article, we'll show how to compress a floppy disk. First, let's look at a couple of reasons why you might want to compress a floppy disk.

Why DoubleSpace a floppy?

We've found a couple of situations in which you might want to compress a floppy disk. For instance, you might need to transfer between computers a file too big to fit on one floppy. A compressed 1.44Mb disk can hold around 2.8Mb of data. So, you can copy larger files to floppy disks without having to back them up. After you copy the files, you can read the data from the disk without having to use the RESTORE command.

If you need to copy several hundred small files to a floppy disk, compressing the disk might help. If each of your files is smaller than the floppy disk's default cluster size (512 bytes), you're probably better off not compressing the floppy disk, because you can put a couple thousand files on the disk anyway. However, if your files are larger than 512 bytes, they'll take up extra clusters; by compressing your floppy disk, you can fit a file as large as 8Kb in a single cluster.

Now you know a few reasons to compress a floppy disk. Before we jump into the technique, let's look at a few more factors to consider.

First, some considerations

There are a few facts you should know about compressing a floppy disk and using it:

- You can compress a floppy disk only from a hard drive that has DoubleSpace installed.
- The floppy disk you want to compress must be formatted, must be larger than 360Kb, and should have at least 650Kb of free space.
- If you want to use a compressed floppy disk on two computers, both computers must be using DoubleSpace.
- You can use DoubleSpace to compress a disk on a DOS 6.0 machine and use that disk on a machine running DOS 6.2—and vice versa.
- If you remove the compressed disk from a DOS 6.0 drive, you'll have to mount the disk before you can use it again. DOS 6.2 has a built-in mounting process.
- If the disk you're compressing isn't blank, you should copy or back up the disk's files before you begin.

We'll address other considerations as we discuss the technique and run through an example.

How to compress a floppy

You can compress a floppy in either of two ways: by using DoubleSpace's menu-oriented interface or by running the DBLSPACE command with switches. The menu interface offers the security of a full help system and screen messages to guide you through the process. Also, it allows you to cancel the process. After you've used the utility a few times, you'll probably want to run DoubleSpace from the command line. To do so, you just need to know the command switches for the options you want to use.

IN THIS ISSUE

Once you compress the disk with either method, you'll want to change the compression ratio. If you want to copy several hundred small text files to the compressed diskette, you won't benefit from DoubleSpace if you retain the default 2-to-1 compression ratio. Instead, you should increase the ratio in order to increase the number of clusters available on your disk. (Read "Why Change the Compression Ratio?" on page 4 for more information on DoubleSpace disk space numbers and compression ratios.) In the following sections, we'll discuss using each method to compress a floppy disk.

Using the menu-oriented interface

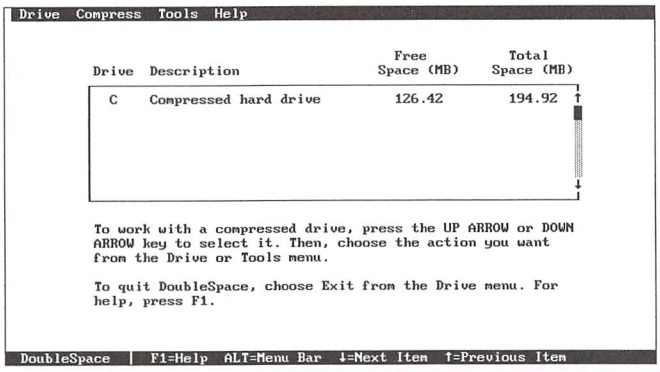
First, let's look at the menu-oriented interface. To start the DoubleSpace utility, place a formatted disk in the floppy drive and enter the command

```
C:\>dblspace
```

DOS loads the DoubleSpace utility and displays the main screen. As you can see in Figure A, the screen shows a menu of options and a list of compressed drives.

At the main screen, press [Alt]C to open the Compress menu, and choose the Existing Drive... command. DoubleSpace checks for drives it can compress and lists the drive that contains your disk. Highlight that entry and press [Enter]. At this point, you're reminded to back up the files on your floppy disk. Press E to Exit and back up your files. Or, press C to Continue and then press C again to start DoubleSpace.

Figure A



The DoubleSpace main screen shows your options and lists the compressed drives.

If you pressed C, DoubleSpace runs the ScanDisk utility to verify the integrity of your disk (in 6.0, it runs CHKDSK instead), compresses the disk, and finally returns to the DoubleSpace main screen. With a blank floppy disk, this takes about a minute; if your disk contains data, it will take a little longer. The main

INSIDE DOS

Inside DOS (ISSN 1049-5320) is published monthly by The Cobb Group.

Prices: Domestic: $49/yr ($6.00 each)
Outside US: $69/yr ($8.50 each)

Phone: Toll free: (800) 223-8720
Local: (502) 491-1900
Customer Relations Fax: (502) 491-8050
Editorial Department Fax: (502) 491-4200

Address: You may address tips, special requests, and other correspondence to

The Editor, *Inside DOS*
9420 Bunsen Parkway, Suite 300
Louisville, Kentucky 40220

For subscriptions, fulfillment questions, and requests for bulk orders, address your letters to

Customer Relations
9420 Bunsen Parkway, Suite 300
Louisville, Kentucky 40220

Postmaster: Second class postage is pending in Louisville, KY. Send address changes to

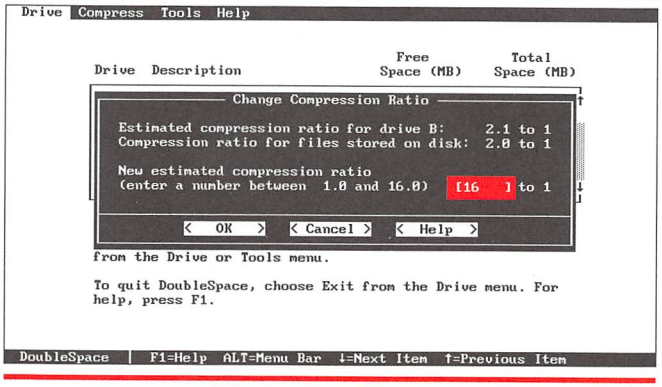
Inside DOS
P.O. Box 35160
Louisville, KY 40232

Authorized Canada Post International Publications Mail (Canadian Distribution) Sales Agreement #XXXXXX CANADA GST #123669673. Send returns to Canadian Direct Mailing Sys. Ltd., 920 Mercer Street, Windsor, Ontario, N9A 7C2. Printed in the USA.

screen now shows that your disk has about twice as much space as it started with.

You also can change the compression ratio. To begin, highlight the listing for your floppy disk and press [Alt]D, which opens the Drive menu. Select Change Ratio... and type *16*—the maximum allowed—in the highlighted text box, as shown in Figure B. Then press [Enter] or click OK. The Total Space column of the main screen now displays the new disk size.

Figure B



You can increase your disk's compression ratio to gain additional storage clusters.

You can view information about your compressed drive by choosing Info... from the Drive menu. You'll see a summary of total space, space used, free space, and the actual and estimated compression ratios. Now you're ready to quit DoubleSpace. DOS automatically mounts the diskette so you can use it immediately. To quit DoubleSpace, select Exit from the Drive menu.

Running DoubleSpace from the DOS prompt

If you want to avoid using the full DoubleSpace utility, you can compress a floppy disk from the DOS prompt by entering the command

```
dblspace drive /compress
```

where *drive* is the letter (with or without the colon) of the drive containing the disk you want to compress. Compressing a disk from the DOS prompt takes about as long as from within the full utility. DoubleSpace returns you to the DOS prompt when the compression process finishes.

Then, you can increase the compression ratio by using the command

```
dblspace drive /ratio=16.0
```

You can use any number from 2.0 to 16.0 as the compression ratio. Use the command

```
dblspace drive /info
```

to view information about your compressed disk. Now, let's use DoubleSpace to compress a floppy disk.

An example

We'll compress a blank 1.44Mb floppy disk in the B: drive. Since we're familiar with the DoubleSpace utility, let's compress the disk from the DOS prompt. Place a formatted 1.44Mb disk in drive B and enter

```
C:\>dblspace b /compress
```

at the prompt. The DoubleSpace utility will automatically begin the process. It will run ScanDisk (CHKDSK in Version 6.0), compress the diskette, and return to the DOS prompt.

Now let's change the compression ratio. Enter

```
C:\>dblspace b /ratio=16.0
```

at the DOS prompt. Once DoubleSpace changes the ratio, you'll see the message shown in Figure C.

Figure C

```
C:\>dblspace b /ratio=16.0
DoubleSpace is examining drive B.
DoubleSpace is changing the compression ratio of drive B.
DoubleSpace is remounting drive B.
The estimated compression ratio of drive B has been changed.
```

You can change the compression ratio from the DOS prompt.

Next, let's view the information about our compressed disk. Figure D on the next page shows the command we entered and its result. Now that we've compressed a floppy disk, let's look at how to use it.

Using a compressed floppy disk

As we mentioned, you can use a diskette DoubleSpace has compressed only on another computer running DoubleSpace. So, be sure to mark the disk as compressed. Diskettes compressed on computers using Versions 6.0 and 6.2 are fully compatible. However, if you place a diskette in a 6.0 drive, you must mount the diskette before you can use it. If you don't, a directory listing will show only the file READTHIS.TXT. When you type this file, you see a message that reminds you to mount the compressed disk before you can use it. (You also see this file when you try to use a compressed diskette on a computer that isn't running DoubleSpace.)

Figure D

```
C:\>dblspace b /info
DoubleSpace is examining drive B.

Compressed drive B is stored on uncompressed drive F in the file
F:\DBLSPACE.000.

    Space used:                0.78 MB
    Compression ratio:         1.4 to 1

    Space free:                3.38 MB
    Est. compression ratio:    16.0 to 1
    Fragmentation:               3%

    Total space:               4.16 MB
```

DBLSPACE /INFO shows you information about your compressed disk.

To mount the floppy disk in a computer using DOS 6.0, place the diskette in the drive, change to that drive, and enter the command

```
B:\>dblspace /mount
```

A computer using DOS 6.2 mounts the disk when you run a command on the floppy disk. Once the disk is mounted, you can use it just like any other diskette.

Uncompressing a floppy disk

Should you want to uncompress a floppy disk, DoubleSpace lets you do that, too. You can't uncompress a full disk—DoubleSpace will tell you how much space you need to free up. In practice, we've found that, if possible, it's better to delete all the files from the disk before you uncompress it.

Why change the compression ratio?

In "Compressing a Floppy Disk with DoubleSpace" starting on page 1, we recommend that you select a 16-to-1 compression ratio when you use DoubleSpace to compress a floppy disk. However, in practice, we discovered that DoubleSpace compresses all files as tightly as it can, no matter what compression ratio we specify. This being the case, you may wonder why you should alter the compression ratio at all.

In fact, you can change only the *estimated* compression ratio—not the *actual* compression ratio. The actual compression ratio reflects the degree to which DoubleSpace flattens files already on the disk. The estimated compression ratio reflects the amount of free space on a disk. When we change the compression ratio, we're just telling DOS to use a different value to estimate the amount of free disk space.

Although we can't tell DoubleSpace how to compress files, changing the compression ratio does let us put more files on a diskette in some circumstances. Table A shows the results of compressing three disks to the default 2-to-1 compression ratio and to a 16-to-1 ratio. As you can see, when you compress the 3.5" HD disk to a 2-to-1 ratio, the capacity doubles and the cluster size jumps to 8,192, but the total number of clusters the disk can hold drops to 338. This means that you can't place more than 338 files on the diskette, no matter how small they are.

However, when you increase the ratio to 16-to-1, the estimated capacity increases to 22.2Mb, and you can place up to 2,711 files on the disk. Another benefit of the high compression ratio on a high-density diskette is that DOS lets you place up to 511 files in its root directory—not the 224 it allows on a normal 1.44Mb disk.

Although a 16-to-1 compression ratio makes more sense when you need to compress a lot of small files, you may also use it when you need to compress larger files. Since DoubleSpace compresses each file at the maximum compression ratio it can handle, large files might end up compressed at a 2-to-1 ratio, even if you specify a greater ratio. So, even here, it's not a bad idea to go ahead and compress the diskette to 16-to-1. You may decide to fill the empty disk space with small files anyway, and a larger ratio would give you more clusters for doing just that.

TABLE A: *Compressing disks at different ratios*

	Capacity	Bytes/ Cluster	Number of Clusters
3.5 HD			
Uncompressed	1.44	512	2847
2:1 ratio	2.8	8192	338
16:1 ratio	22.2	8192	2711
3.5 LD			
Uncompressed	730K	1024	713
2:1 ratio	1.3	8192	162
16:1 ratio	10.6	8192	1301
5.25 HD			
Uncompressed	1.2	512	2371
2:1 ratio	2.3	8192	279
16:1 ratio	18.25	8192	2235

To use the menu-driven utility, enter

```
C:\>dblspace
```

at the prompt. Then, at the main screen, highlight the disk you want to restore to normal, press [Alt]T to open the Tools menu, and select the Uncompress... command. When you answer Yes to the Uncompress Confirmation message, DoubleSpace runs ScanDisk (CHKDSK on a 6.0 machine) and then uncompresses the disk.

To avoid the menus and prompts, you can uncompress the disk at the command line. Just enter the command

```
C:\>dblspace drive /uncompress
```

and DoubleSpace will uncompress the disk and return you to the DOS prompt.

When DoubleSpace uncompresses a disk, it moves the READTHIS.TXT file from the host drive to the disk you're uncompressing. After you uncompress the disk, you can delete the READTHIS.TXT file. Now the floppy is back to its original state.

Conclusion

A floppy disk compressed with DoubleSpace has roughly twice the storage space of an uncompressed disk. If you want to store files that are too large for a normal diskette, or if you want to store more than a thousand medium-sized (1Kb to 8Kb) files, you should use the DoubleSpace techniques we discussed in this article to compress your floppy disk.

SECURITY TIP

VERSION 5.0 & 6.x

Safeguarding your system files

Are you one of the many DOS users who regularly, if accidentally, wipes out your root directory? If so, we have a tip for you: Hide your system files. By hiding your system files, you protect them from accidental deletion, and you get added benefits—you'll still be able to edit the files and your root directory will look neater. In this article, we'll show you how to hide the AUTOEXEC.BAT, CONFIG.SYS, and COMMAND.COM files.

Hiding your system files

To hide the system files, we'll attach the ATTRIB command's hidden attribute to them. DOS introduced the hidden attribute in Version 5. Hidden files remain in the directory but do not appear in the directory listing. You can't delete, copy, back up, move (DOS 6), print, rename, or use the XCOPY command with hidden files. However, you can type and edit hidden files. Also, you can delete hidden files with DOS 6's DELTREE command.

The ATTRIB command uses the syntax

```
ATTRIB +h filename
```

In this command, the plus sign (+) indicates you want to add an attribute—h (for hidden) in this case; *filename* is the name of the file you want to hide. You can also use wildcard characters to set the attributes of a group of files.

To double-check that the file is hidden, enter the ATTRIB command followed by the name of the file you want to check, for example

```
C:>attrib autoexec.bat
```

If you want to check the attributes of all the files in the current directory, just type *attrib* and press [Enter]. Alternatively, you can use the attribute switch with a directory command to list files that have specific attributes. For example, if you enter the command

```
C:>dir /a:h
```

you'll see a regular directory listing of all the files in the current directory that have the hidden attribute.

To remove an attribute, simply use a minus sign (-) where you previously used the plus sign to add the attribute. Now, let's use ATTRIB to hide the system files in your root directory.

Let's hide those files

To protect your system, hide the AUTOEXEC.BAT, CONFIG.SYS, and COMMAND.COM files. We'll have to set the attributes of our files individually. Since

their names have no common and unique characters, we can't use wildcards. At the DOS prompt, enter the following commands in turn:

```
C:>attrib +h autoexec.bat
C:>attrib +h config.sys
C:>attrib +h command.com
```

It's that simple. At this point, you can hide any other important files you don't want to lose inadvertently.

Let's look for those files

Now let's verify that our files are actually hidden. First, enter a simple directory command; you shouldn't see the three files listed. Next, check the attributes of all the files in the directory by entering the command *attrib*. You should see a listing similar to

```
HR    C:\IO.SYS
HR    C:\MSDOS.SYS
H     C:\COMMAND.COM
H     C:\AUTOEXEC.BAT
      C:\CONFIG.OLD
      C:\VIRSIGS.MS
 R    C:\WINA20.386
H     C:\CONFIG.SYS
```

Next, let's use the attribute switch with the directory command to view only the files that carry the hidden attribute. The command and its output are shown at the top of the next column. As you can see, the listing also displays the IO.SYS and MSDOS.SYS files, which DOS installed as read-only, hidden files. (Your listing might include IBMBIO.COM and IBMDOS.COM instead.)

```
C:\>dir /a:h

 Volume in drive C is HOST_FOR_C
 Volume Serial Number is 16D9-1337
 Directory of C:\

IO         SYS     40566  09-30-93   6:20a
MSDOS      SYS     38138  09-30-93   6:20a
COMMAND    COM     54619  09-30-93   6:20a
AUTOEXEC   BAT       183  11-24-93   1:58p
CONFIG     SYS       227  11-15-93   7:11p
```

Why not use the system attribute?

DOS 5 also introduced the system attribute, which works just like the hidden attribute in most cases. You might wonder why we didn't use the system attribute to hide our system files. We could have attached the system attribute (+s) to the files, but we've found that you can't execute a command that's marked with the system attribute from the command line or from a batch file. If, for example, your AUTOEXEC.BAT file carries the system attribute, you won't be able to run it from the DOS prompt or from a batch file.

Conclusion

If you want to protect your system files, just attach the ATTRIB command's hidden attribute to them. You'll still be able to view and edit them, but you won't be able to accidentally delete them. In this article, we showed you how to hide your system files.

VAN WOLVERTON

VERSION 5.0 & 6.x

Personalizing your keyboard with shortcut keys

In the December 1993 article "Creating Your Own DOS Command Shortcuts," we looked at how you can use the PROMPT command to define your own shortcut keys so you can run a command with one combination keystroke, such as [Ctrl]A or [Alt][F10]. This technique lets you use one shortcut key to run DOS commands, batch commands, DOSKEY macros, or any other program you can run from the command prompt. The December article included a table of key codes and extended key codes for identifying the key you want to redefine.

In this article, we'll show you several batch files you can use to easily redefine your keys. (We use the word *key* to mean either a single key, such as [F1], or a key combination, such as [Ctrl]P.)

The PROMPT technique

Let's quickly recap the technique. You use the PROMPT command to send an ANSI.SYS command to the console (the device name DOS uses to describe the display plus the keyboard). The ANSI.SYS command that redefines a key begins with the Escape character

(ASCII character 27) and includes both the code that identifies the key and the new result of pressing the key.

You use the PROMPT command to send the ANSI.SYS command because it lets you represent an Escape character with *$e*. You can't just press the [Escape] key because that tells DOS to cancel whatever you've typed.

Here's a short example that shows how the technique works. It also might prove useful to those of you who use a Laserjet or Laserjet-compatible printer.

If you copy the console or redirect the output of a command to a Hewlett-Packard Laserjet or a laser printer compatible with the Laserjet series, the printer doesn't print anything until you send it a Form Feed character. This lack of response is disconcerting until you figure out what's going on. Although the printer appears to be malfunctioning, it's simply waiting to be told that it has reached the end of a page.

The Form Feed character is ASCII character 12, also called [Ctrl]L because you can create it by holding down the [Ctrl] key and pressing L (which DOS echoes as <^L>). This character tells a printer that the end of a page has been reached, so it's time to print and eject the current page and start another. You can send a Form Feed character to the printer by using an ECHO command. Assuming your printer is attached to LPT1, you'd use the command

```
echo ^L > lpt1
```

(Remember, you create the ^L in the command by pressing [Ctrl]L.)

If you have to send a form feed to your printer several times, there's an easier way than typing several ECHO commands: You can redefine a key combination to type the ECHO command for you. Here's a PROMPT command that redefines [Alt]P (its extended key code is 0;25) to send a form feed ([Ctrl]L) to LPT1:

```
prompt $e[0;25;"echo ^L > prn";13p
```

Now, whenever you send to the printer some data that doesn't include a form feed, just press [Alt]P. If you do this fairly often, put the PROMPT command in your AUTOEXEC.BAT file. In most instances, this redefinition won't affect your normal use of the computer. And remember, you'll have to redefine your regular command prompt after entering this PROMPT command.

Redefining a key with a batch file

Saving the prompt definition, typing a PROMPT command to redefine a key, and then restoring the prompt definition is repetitious and time consuming—the perfect situation for a batch file. NEWKEY.BAT takes care of most of the tedium involved in redefining a key. It lets you redefine any key, whether it has a one-digit key code or an extended key code (0 followed by a semicolon and another number).

NEWKEY.BAT redefines a key so that it produces a character string followed by a carriage return (the carriage return means you won't have to press [Enter] after pressing the shortcut key). You invoke this batch command using two parameters: the key code or extended key code of the key to redefine and the character string you want to produce by pressing the key. Type in the code in Figure A and save it as NEWKEY.BAT.

Figure A

```
@echo off
set psav=%prompt%
if %1==0 goto EXTENDED
prompt $e[%1;"%2";13p
goto END

:EXTENDED
prompt $e[0;%2;"%3";13p

:END
@echo on
prompt=%psav%
@echo off
@set psav=
```

NEWKEY.BAT lets you redefine any key.

The first SET command in NEWKEY.BAT makes a copy of the current prompt definition in an environment string named PSAV. The IF command checks to see if the first parameter is 0, which means you specified an extended key code. If so, NEWKEY.BAT transfers control to the label EXTENDED.

If the first parameter you typed with the batch command isn't 0, the PROMPT command in the next line sends an ANSI.SYS command to redefine a key. The first parameter is the key code of the key you're redefining and the second parameter is the new result of pressing that key. The ;13 following the closing quotation mark puts a carriage return at the end so you don't have to press [Enter] after pressing the redefined shortcut key. NEWKEY.BAT then passes control to the label END.

The batch file carries out the PROMPT command following the label EXTENDED if you use two parameters when you invoke the batch command. It accomplishes the same purpose as the earlier

PROMPT command, but this one must make allowance for the two numbers in the extended key code (the first of which is always 0). Hence it uses three replaceable parameters: The first (%1) is the first extended key code number (0), the second (%2) is the second extended key code number, and the third (%3) is the redefined result of pressing the key identified by %1 and %2. Again, it adds the carriage return at the end (;13) so you don't have to press [Enter] after pressing the shortcut key.

NEWKEY.BAT carries out the two commands following the label END whether the key has a one-digit key code or a two-digit extended key code. The PROMPT command restores the original prompt definition by copying it from the environment string PSAV. The SET command deletes the environment string PSAV because it's no longer needed.

The ECHO ON and ECHO OFF commands around the final PROMPT command are necessary because the key redefinition won't work unless this PROMPT command is echoed. You can accomplish the same task by beginning each command in the batch file except the final PROMPT command with an @ sign, as you'll see in a moment.

Using NEWKEY

With NEWKEY.BAT, you can redefine a key with one simple command. To redefine [Ctrl]S to clear the screen, for example, type the following:

```
C:\>newkey 19 cls
```

Now, whenever you want to clear the screen, press [Ctrl]S.

This technique works only if you want to redefine a key to produce a single character string (*cls*, for example, or *edit*). If you want to redefine a key to carry out a command that requires more than one character string, such as *dir /on /p*, put the command in a batch file or a DOSKEY macro and redefine the key to produce the name of the batch command or macro.

If you want to carry out a command that includes redirection (< or >) or pipe (¦) characters, you'll have to use the special character combinations DOSKEY includes for specifying these characters. DOS won't carry out the intended function if you use the actual redirection or pipe characters in a batch file or DOSKEY macro. The special character combinations are:

```
$g>  Redirect output
$l<  Redirect input
$b¦  Pipe output to a command
```

So, to use NEWKEY.BAT to redefine [Alt]P to send a form feed to the printer, type the following two commands:

```
C:\>doskey SendFormFeed=echo ^L $g lpt1
C:\>newkey 0;25 SendFormFeed
```

After you do so, press [Alt]P to print pending output from your printer.

Restoring a key with a batch file

The ANSI.SYS command that restores the original meaning of a key simply specifies the key's identifying code twice. This simplifies the task a bit, because we don't have to worry about specifying a new definition. Figure B shows the contents of RESTKEY.BAT, which restores the original meaning of a key. It takes only one parameter: the key code or extended key code of the key to be restored.

Figure B

```
@set psav=%prompt%
@if %1==0 goto EXTENDED
@prompt $e[%1;%1p
@goto END

:EXTENDED
@prompt $e[%1;%2;%1;%2p

:END
prompt=%psav%
@set psav=
```

RESTKEY.BAT restores standard key definitions.

RESTKEY.BAT is almost a carbon copy of NEWKEY.BAT, except it uses the @ character to suppress echoing the commands rather than the ECHO command used in NEWKEY.BAT. The only functional difference is in the first two PROMPT commands that send the ANSI.SYS command to the console. In NEWKEY.BAT, these commands used the second parameter typed with the command—the redefined result of pressing the key—as the second parameter in the ANSI.SYS command. But you type only one parameter with RESTKEY.BAT—the key's identifying code—which RESTKEY.BAT uses as both the first and second parameter of the ANSI.SYS command to restore a key's original meaning.

To use RESTKEY.BAT, suppose you redefined [Ctrl]S to clear the screen as in the preceding example. You would restore the original meaning of [Ctrl]S by typing:

```
C:\>restkey 19
```

Similarly, if you redefined [Alt]P to carry out the SendFormFeed DOSKEY macro, the following command would restore the original meaning of [Alt]P:

```
C:\>restkey 0;25
```

Setting up a series of shortcuts

If you routinely define a series of shortcut keys, you could simply put a series of CALL NEWKEY commands in AUTOEXEC.BAT. But you may want more than one set of definitions—for example, you might want alternate sets of definitions for keys [F1] through [F4], depending on the work you're doing.

Again, DOS batch files provide the solution. Figure C shows two batch files (named NEW1.BAT and NEW2.BAT) that define [F1] through [F4] to run two different sets of programs. The third batch file, named RESTALL.BAT, restores the original definitions of [F1] through [F4].

Figure C

```
rem NEW1.BAT
   @call newkey 0;59 help
   @call newkey 0;60 edit
   @call newkey 0;61 cls
   @call newkey 0;62 123

rem NEW2.BAT
   @call newkey 0;59 help
   @call newkey 0;60 dosshell
   @call newkey 0;61 qbasic
   @call newkey 0;62 restall

rem RESTALL.BAT
   @call restkey 0;59
   @call restkey 0;60
   @call restkey 0;61
   @call restkey 0;62
```

You use batch files to define different sets of commands.

There's nothing tricky here. NEW1.BAT and NEW2.BAT each call NEWKEY.BAT four times to redefine keys [F1] through [F4]. Both sets define [F1] to display help; the final redefinition in NEW1.BAT causes pressing [F4] to produce the string *123*. This string could be the name of a macro or batch file that carries out more than one command to change to the directory that contains 123 and starts the program.

RESTALL.BAT simply calls RESTKEY.BAT four times to restore the original definitions of [F1] through [F4]. To make things a bit more convenient, the final redefinition in NEW2.BAT makes pressing [F4] carry out RESTALL.BAT. This lets you use a shortcut key to restore the original key meanings.

If you spend a bit of time setting up these batch files, you can really speed up routine tasks such as printing directories, starting programs, and the like. By combining this technique with your batch files and DOSKEY macros, you can personalize your keyboard as much as you want and simplify the housekeeping DOS requires.

Placing system files on a boot disk

In "Creating a DOS 6 Emergency Boot Disk" in the October 1993 issue, we told you to format your emergency boot disk by issuing the command

```
C:\>format a: /f:n /s
```

where *n* is the capacity of your disk. We then told you to transfer the system files onto the disk using the SYS command:

```
C:\>sys c:\ a:
```

However, the FORMAT command's /S switch transfers the system files to the disk. Therefore, you don't need to issue the SYS command when you use the /S switch with FORMAT. We apologize for any confusion this error might have caused and thank the readers who pointed out our mistake.

VERSION 5.0 & 6.x

APPEND makes accessing data files easy—but is it worth the risks?

What the PATH command does for executable files, the APPEND command does for data files: APPEND establishes a list of directories for DOS to search when it's looking for nonexecutable files. The major benefit of using PATH and APPEND is that you can store program files and data files in separate directories but run the program and access the data files from *any* directory without typing or trying to remember long pathnames. These two commands can be real timesavers, but as we'll explain in this article, trying to save time with APPEND might create confusion later.

APPEND basics

Before we look at the risks of using APPEND, let's review the basics. As we mentioned, APPEND sets up a search path to enable DOS to locate nonexecutable files. DOS treats the list of directories as an extension of the current directory so it can access data files without knowing the path.

APPEND is actually a memory-resident program (TSR) you can load at the command prompt or in a batch file. To create APPEND's directory list, you use the syntax

```
APPEND path1;path2;...pathx
```

where *path1*; *path2*; ...*pathx* represents the list of directories you want DOS to search. Once you've established APPEND's directory list, you can check its contents by issuing the APPEND command without any parameters or switches. To eliminate APPEND's directory list, you use the command

```
APPEND ;
```

You can also load APPEND with some optional switches. The /E switch creates the APPEND environment variable, which stores the directory list as a character string. If you want to create the APPEND environment variable, you must issue the command

```
APPEND /E
```

without any path specifications the first time you load APPEND. Once you've issued this command, you issue the APPEND command *with* path specifications to create the directory list.

Now, suppose you want DOS to check APPEND's directories for files whose specifications include an incorrect path. To enable this feature, you include the /PATH:ON switch when you issue the APPEND command. To toggle off this feature once you've turned it on, you issue the APPEND command again but include the /PATH:OFF switch instead.

Another toggle switch you can use with APPEND is /X. By default, DOS checks APPEND's directory list only when an application makes a request to open a file. By running APPEND with the /X:ON switch, you enable DOS to check the directory list when an application executes a file search or wants to run another program. The /X:OFF switch returns APPEND to the default.

An example

Let's say you want to create the APPEND environment variable and you want to list the C:\WORD\DOCS, C:\123\SPREADS, and C:\DB\DATA directories in it. You also want DOS to check APPEND's list when it encounters an incorrect path specification and when an application executes a file search. To load APPEND to meet these requirements, you'd issue the commands

```
C:\>append /e
C:\>append c:\word\docs;c:\123\spreads;c:\db\data
    /path:on /x:on
```

Suppose you then decide you don't want DOS to check the APPEND list when a file request includes a path specification. To disable this feature, you'd issue the command

```
C:\>append /path:off
```

Say you also decide you want DOS to search APPEND's list only when a program makes an open file request. To make this change, you'd issue the command

```
C:\>append /x:off
```

Finally, to suspend APPEND's directory list, you'd issue the command

```
C:\>append ;
```

Alternatively, you can use the command

```
C:\>set append=
```

to nullify the directory list, since you created the APPEND environment variable by loading APPEND with the /E switch.

APPEND risks

Now that you're familiar with how APPEND works, let's examine the risks of using it. The most important one to watch out for involves programs, such as word processors, that modify a *copy* of a file when you input changes instead of making changes directly to the original.

For instance, suppose you've set up the APPEND directory list in the example on the previous page:

```
append c:\word\docs;c:\123\spreads;c:\db\data
```

where C:\WORD\DOCS is the directory containing all your word processing documents. Next, assume that C:\ is the current directory, your word processing program is running, and you've opened a document to modify. Since the directory containing the document appears in APPEND's list, DOS opened a copy of the document without changing the current directory. Therefore, when you save the changes, the program will write them to the copy of the document in the current directory, *not* to the original document in the C:\WORD\DOCS directory.

If you use Microsoft Windows, you run two other risks when you use APPEND. First, since APPEND is a memory-resident program, it uses memory (and a lot of it), thereby reducing the amount of memory available to Windows. Second, by treating the directories in its list as an extension of the current directory, APPEND can confuse Windows about the true location of files.

If you use the ASSIGN command to reassign a drive letter to a different drive, you run even more risks by using APPEND. You must make sure you load APPEND before issuing the ASSIGN command; otherwise, DOS won't know where to look for the files. In addition, you can't include directories on reassigned drives in APPEND's directory list.

Finally, APPEND can cause a lot of confusion on networks. Including directories on network drives in APPEND's directory list will cause trouble. And if you use IBM's PC LAN, you're sure to encounter problems—PC LAN contains an unrelated program also called APPEND.

File organization—a safer alternative

With all the risks involved in using APPEND, the convenience of having easy access to data files hardly seems worth it. By building an organized directory tree structure and storing files in their appropriate directories, you eliminate the need to create a search path for DOS since you'll always know exactly where to find the file you're looking for. If you still want easy access to data files while you're running programs, use the PATH command to establish a directory list for DOS to locate executable files, then switch to the directory containing a program's data files before you run the program.

LETTERS

ECHOing redirection characters in a batch file

I write a lot of batch files that contain instructions for users—usage notes, error messages, and the like. I'm having trouble echoing the pipe character to the screen in a message. My code reads

```
:ONERROR
ECHO ==== More than 1 screen of entries
ECHO ==== Use syntax: GET %1 | MORE
GOTO END
```

The batch file echoes the second ECHO line up to the %1 parameter and stops. Can you help me? I have to be able to tell my users exactly what they must type at the command line to use the batch file, and the ECHO command doesn't seem to work.

Darren McGee
Louisville, Kentucky

Mr. McGee has found one of the limits of using the ECHO command in a batch file. You can use ECHO to type messages to the screen. You also can use ECHO to direct text into a filename or as input to another command. For example, you can use the pipe operator in the command *echo 02-07-94 | date* to change your system date.

Also, you can use the command *echo xxx > ex.doc* to create a file named EX.DOC that contains the text *xxx*.

If you use a redirection character in a batch file—even after the ECHO command—DOS tries to execute it, and you can get unexpected results.

In Mr. McGee's case, the batch file is trying to pipe the entire left side of the second ECHO command through the MORE command. Since the command is only one line long, MORE has no effect.

To echo a message that contains redirection characters, you can create a text file that contains the message and then type that file from within the batch file. For example, you can create a text file called GET.DAT that contains the lines

```
==== More than 1 screen of entries
==== Use syntax: GET %1 | MORE
```

Then, you can replace the two ECHO commands in the :ONERROR routine with the command *TYPE GET.DAT*. The next time the batch file runs the :ONERROR routine, it will display the message, including the pipe operator. This solution isn't elegant and it may slow a batch file's execution, but it works.

The root directory has limits

I feel you have given Mr. Greg Geis some bad information regarding XCOPY not being able to copy more than 224 files at one time ("XCOPY Has Its Limits," November 1993). Mr. Geis's problem most likely stems from the fact that a 1.44Mb floppy can accept only 224 entries in the root directory.

To test XCOPY, I copied all *.FOT and *.TTF files from my \WINDOWS directory to an empty one and used XCOPY to transfer them to another empty directory. It quickly copied 386 files totaling 10.4Mb. Thus, XCOPY itself probably has no practical file number or byte limit.

Had Mr. Geis made a subdirectory on his 1.44Mb diskette and used XCOPY to copy his 438 files there, it would have worked if the files didn't exceed the storage capacity of the floppy.

Only root directories are limited on entries. There is no limit to the number of files (except disk storage capacity) that subdirectories can handle.

Floyd O'Neal
Olympia, Washington

Mr. O'Neal has certainly gotten to the root of the problem we reported in the November article—it isn't the XCOPY command that has limits, it's the root directory. DOS indeed limits the number of entries—any combination of filenames and subdirectories—in the root directory of any disk. To understand these limits, we need to look at how DOS uses the root directory.

The root directory appears to be a list of files and directories. But it's actually a table consisting of 32-byte entries that store a file's name, extension, attribute, date and time of creation, size, and starting cluster address. The starting cluster address links the file's root directory entry to its starting entry in the file allocation table (FAT). The FAT keeps track of how each disk cluster is being used.

When DOS puts a file into a cluster, it places the file's beginning address in the root directory. At the same time, DOS stores in the FAT entry either the address of the next part of the file or the end-of-file code. So the root directory acts as a name index to the FAT, which is the doorway to DOS's address-based storage system.

When you format a hard or floppy disk, DOS reserves several sectors at the beginning of the disk for boot information, the FAT, and the root directory. DOS reserves from 4 to 32 sectors for the root directory, based on the size of the disk. Since the root directory must fit into a specific area, it's easy to see why DOS limits the number of entries in the root directory.

How many files will it hold?

Determining how many files you can place in a root directory is a simple mathematical matter. Each sector contains 512 bytes, so you can calculate the total space available for directory information by multiplying 512 by the number of reserved sectors on the disk. To determine how many root directory entries DOS allows, simply divide the total space available for directory information by 32—the number of bytes each directory entry takes up.

Table A summarizes the number of sectors DOS reserves for root directory information and the maximum number of files and directories DOS allows in the root directories of disks of various sizes.

TABLE A: *Sectors and root directory entries, by disk size*

Size of disk	Number of Sectors	Number of Entries
160Kb, 180Kb	4	64
320Kb, 360Kb	7	112
720Kb	7	112
1.2Mb, 1.44Mb	14	224
2.88Mb	15	240
hard disk	32	512